Parsing

Formele en natuurlijke talen

Lecture 13

Goals for today

- **Conceptualize** what parsing is for a (context-free) grammar
- **Describe** the conceptual steps involved in several different parsing algorithms: top-down, bottom-up, left corner, CYK
- **Apply** these algorithms to actual grammars
- **Compare** parsers in terms of complexity and efficiency to understand their pros and cons

What is parsing?

How to tell whether a particular string s belongs to a particular context-free language $\mathcal{L}$ (characterized by grammar $\mathcal{G}$?

What is parsing?

How to tell whether a particular string s belongs to a particular context-free language $\mathcal{L}$ (characterized by grammar $\mathcal{G}$?

Informally:

- If $s \in \mathcal{L}$, there is some finite sequence of applications of production rules of $\mathcal{G}$ starting from the start symbol and ending in s

What is parsing?

How to tell whether a particular string s belongs to a particular context-free language $\mathcal{L}$ (characterized by grammar $\mathcal{G}$?

Informally:

- If $s \in \mathcal{L}$, there is some finite sequence of applications of production rules of $\mathcal{G}$ starting from the start symbol and ending in s

Derivations involve lots of choices! So we need a systematic way to check whether such a derivation exists: **parsing**

A CFG for (a fragment of) Dutch

$$
\begin{aligned}
\text{S} &\rightarrow \text{DP VP} \\
\text{S} &\rightarrow \text{V DP DP} \\
\text{S} &\rightarrow \text{V DP Adv DP} \\
\text{DP} &\rightarrow \text{D NP} \\
\text{NP} &\rightarrow \text{N} \\
\text{NP} &\rightarrow \text{A N} \\
\text{VP} &\rightarrow \text{V DP} \\
\text{VP} &\rightarrow \text{V Adv DP} \\
\text{V} &\rightarrow \text{bakt} \\
\text{D} &\rightarrow \text{de} \mid \text{een} \\
\text{N} &\rightarrow \text{man} \mid \text{taart} \\
\text{Adv} &\rightarrow \text{morgen} \\
\text{A} &\rightarrow \text{lekkere}
\end{aligned}
$$

A CFG for (a fragment of) Dutch

$S \to DP\ VP$
$S \to V\ DP\ DP$
$S \to V\ DP\ Adv\ DP$
$DP \to D\ NP$
$NP \to N$
$NP \to A\ N$
$VP \to V\ DP$
$VP \to V\ Adv\ DP$
$V \to$ bakt
$D \to$ de | een
$N \to$ man | taart
$Adv \to$ morgen
$A \to$ lekkere

derivation of bakt de man morgen een lekkere taart	
S	
V DP Adv DP	S → V DP Adv DP
bakt DP Adv DP	V → bakt
bakt D NP Adv DP	DP → D NP
bakt de NP Adv DP	D → de
bakt de N Adv DP	NP → N
bakt de man Adv DP	N → man
bakt de man morgen DP	Adv → morgen
bakt de man morgen D NP	DP → D NP
bakt de man morgen een NP	D → een
bakt de man morgen een A N	NP → A N
bakt de man morgen een lekkere N	A → lekkere
bakt de man morgen een lekker taart	N → taart

Why parse?

Problem: Making choices is hard:

$$\begin{aligned} NP &\rightarrow N \\ NP &\rightarrow A\ N \end{aligned}$$

Why parse?

Problem: Making choices is hard:

$$\begin{aligned} NP &\rightarrow N \\ NP &\rightarrow A\ N \end{aligned}$$

If we make the wrong choice in a derivation, we may fail to accept a string that would otherwise be accepted

Why parse?

Problem: Making choices is hard:

$$\begin{array}{lcl} \mathrm{NP} & \rightarrow & \mathrm{N} \\ \mathrm{NP} & \rightarrow & \mathrm{A\ N} \end{array}$$

If we make the wrong choice in a derivation, we may fail to accept a string that would otherwise be accepted

So, we need some procedure to make sure that we consider **all** possible (relevant) choices

Formal, explicit parsing methods

Still widely used for a variety of applications!

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden en

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden en katten

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden en katten ze

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden en katten ze dus

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden en katten ze dus vaak

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden en katten ze dus vaak af

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden en katten ze dus vaak af

De

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden en katten ze dus vaak af

De oude

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden en katten ze dus vaak af

De oude stal

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden en katten ze dus vaak af

De oude stal van

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden en katten ze dus vaak af

De oude stal van de

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden en katten ze dus vaak af

De oude stal van de dominee

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden en katten ze dus vaak af

De oude stal van de dominee maar

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden en katten ze dus vaak af

De oude stal van de dominee maar werd

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:
  `function` $\rightarrow$ `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden en katten ze dus vaak af

De oude stal van de dominee maar werd gepakt

Formal, explicit parsing methods

Still widely used for a variety of applications!

- Programming language syntax is (partly) a CFG:
- Python:

  `function` → `def NAME ( PARAMS ): DOCSTRING BODY`
- Parsing makes interpreting Python (etc) possible for a computer
- Automatic structual annotation of text
- Human language comprehension: incremental parsing

Mijn zussen haten honden en katten ze dus vaak af

De oude stal van de dominee maar werd gepakt
'garden path' sentence (*intuinzin*, h/t Klaas Seinhorst & Daan Wesselink)

Kinds of parsing algorithms

Top-down parsing: Start from top symbol and work down to s

Top-down parsing

Top-down parsing: Start from top symbol and work down to s

$\boxed{\text{parsing the DP een lekkere taart}}$

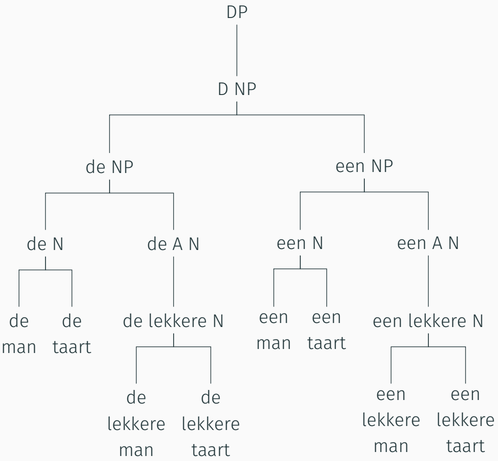
Top-down parsing: Start from top symbol and work down to s

parsing the DP een lekkere taart

DP $\rightarrow$ D NP
NP $\rightarrow$ N
NP $\rightarrow$ A N
V $\rightarrow$ bakt
D $\rightarrow$ de | een
N $\rightarrow$ man | taart
A $\rightarrow$ lekkere

Top-down parsing: Start from top symbol and work down to s

parsing the DP een lekkere taart



DP $\rightarrow$ D NP
NP $\rightarrow$ N
NP $\rightarrow$ A N
V $\rightarrow$ bakt
D $\rightarrow$ de | een
N $\rightarrow$ man | taart
A $\rightarrow$ lekkere

Concrete top-down algorithm: LL-parsing

LL = from **L**eft to Right, **L**eftmost derivation

Concrete top-down algorithm: LL-parsing

LL = from Left to Right, Leftmost derivation

1. Begin with a stack containing only $

Concrete top-down algorithm: LL-parsing

LL = from **L**eft to Right, **L**eftmost derivation

1. Begin with a stack containing only \$
2. Push the start symbol on top the stack

Concrete top-down algorithm: LL-parsing

LL = from **L**eft to Right, **L**eftmost derivation

1. Begin with a stack containing only \$
2. Push the start symbol on top the stack
3. Do one of the following:
   a) **predict**: Take the top element of the stack α and replace it with β if $\alpha \to \beta$ is a production rule
   b) **match**: Take the top element of the stack α and remove it if α is the next symbol in the input. Then read α.

Concrete top-down algorithm: LL-parsing

LL = from Left to Right, Leftmost derivation

1. Begin with a stack containing only \$
2. Push the start symbol on top the stack
3. Do one of the following:
   a) **predict**: Take the top element of the stack α and replace it with β if $\alpha \to \beta$ is a production rule
   b) **match**: Take the top element of the stack α and remove it if α is the next symbol in the input. Then read α.

- Repeat step 3 until there is no more input to read and the stack contains only \$.
- If derivation fails, **backtrack** to the most recent choice point and make a different choice that you have not already tried.

LL-parsing: example

$S \to X0$

$S \to X1$

$X \to 1$

$X \to 0X$

LL-parsing: example

input	stack
0011	$\$$

S $\rightarrow$ X0

S $\rightarrow$ X1

X $\rightarrow$ 1

X $\rightarrow$ 0X

LL-parsing: example

input	stack
0011	$\$$
0011	$S\$$

S $\rightarrow$ X0

S $\rightarrow$ X1

X $\rightarrow$ 1

X $\rightarrow$ 0X

LL-parsing: example

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$

S $\rightarrow$ X0

S $\rightarrow$ X1

X $\rightarrow$ 1

X $\rightarrow$ 0X

LL-parsing: example

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$

backtrack

S $\rightarrow$ X0

S $\rightarrow$ X1

X $\rightarrow$ 1

X $\rightarrow$ 0X

LL-parsing: example

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$

backtrack

0011	$0X0\$$

S $\rightarrow$ X0

S $\rightarrow$ X1

X $\rightarrow$ 1

X $\rightarrow$ 0X

LL-parsing: example

S → X0

S → X1

X → 1

X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$

backtrack

0011	$0X0\$$
011	$X0\$$

LL-parsing: example

S $\rightarrow$ X0

S $\rightarrow$ X1

X $\rightarrow$ 1

X $\rightarrow$ 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$
backtrack	
0011	$0X0\$$
011	$X0\$$
011	$10\$$
backtrack	

LL-parsing: example

S → X0

S → X1

X → 1

X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$
backtrack	
0011	$0X0\$$
011	$X0\$$
011	$10\$$
backtrack	
011	$0X0\$$

LL-parsing: example

S → X0

S → X1

X → 1

X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$
backtrack	
0011	$0X0\$$
011	$X0\$$
011	$10\$$
backtrack	
011	$0X0\$$
11	$X0\$$

LL-parsing: example

S → X0

S → X1

X → 1

X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$

backtrack

input	stack
0011	$0X0\$$
011	$X0\$$
011	$10\$$

backtrack

input	stack
011	$0X0\$$
11	$X0\$$
11	$10\$$

LL-parsing: example

S → X0

S → X1

X → 1

X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$
backtrack	
0011	$0X0\$$
011	$X0\$$
011	$10\$$
backtrack	
011	$0X0\$$
11	$X0\$$
11	$10\$$
1	$0\$$
backtrack	

LL-parsing: example

S → X0

S → X1

X → 1

X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$
backtrack	
0011	$0X0\$$
011	$X0\$$
011	$10\$$
backtrack	
011	$0X0\$$
11	$X0\$$
11	$10\$$
1	$0\$$
backtrack	
0011	$X1\$$

LL-parsing: example

S → X0
S → X1
X → 1
X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$

backtrack

0011	$0X0\$$
011	$X0\$$
011	$10\$$

backtrack

011	$0X0\$$
11	$X0\$$
11	$10\$$
1	$0\$$

backtrack

0011	$X1\$$

input	stack
0011	$11\$$

backtrack

LL-parsing: example

S → X0
S → X1
X → 1
X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$
backtrack	
0011	$0X0\$$
011	$X0\$$
011	$10\$$
backtrack	
011	$0X0\$$
11	$X0\$$
11	$10\$$
1	$0\$$
backtrack	
0011	$X1\$$

input	stack
0011	$11\$$
backtrack	
0011	$0X1\$$

LL-parsing: example

S → X0

S → X1

X → 1

X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$
backtrack	
0011	$0X0\$$
011	$X0\$$
011	$10\$$
backtrack	
011	$0X0\$$
11	$X0\$$
11	$10\$$
1	$0\$$
backtrack	
0011	$X1\$$

input	stack
0011	$11\$$
backtrack	
0011	$0X1\$$
011	$X1\$$

LL-parsing: example

S → X0

S → X1

X → 1

X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$
backtrack	
0011	$0X0\$$
011	$X0\$$
011	$10\$$
backtrack	
011	$0X0\$$
11	$X0\$$
11	$10\$$
1	$0\$$
backtrack	
0011	$X1\$$

input	stack
0011	$11\$$
backtrack	
0011	$0X1\$$
011	$X1\$$
011	$11\$$
backtrack	

LL-parsing: example

S → X0

S → X1

X → 1

X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$

backtrack

0011	$0X0\$$
011	$X0\$$
011	$10\$$

backtrack

011	$0X0\$$
11	$X0\$$
11	$10\$$
1	$0\$$

backtrack

0011	$X1\$$

input	stack
0011	$11\$$

backtrack

0011	$0X1\$$
011	$X1\$$
011	$11\$$

backtrack

011	$0X1\$$

LL-parsing: example

S → X0

S → X1

X → 1

X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$

backtrack

0011	$0X0\$$
011	$X0\$$
011	$10\$$

backtrack

011	$0X0\$$
11	$X0\$$
11	$10\$$
1	$0\$$

backtrack

0011	$X1\$$

input	stack
0011	$11\$$

backtrack

0011	$0X1\$$
011	$X1\$$
011	$11\$$

backtrack

011	$0X1\$$
11	$X1\$$

LL-parsing: example

S → X0
S → X1
X → 1
X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$
backtrack	
0011	$0X0\$$
011	$X0\$$
011	$10\$$
backtrack	
011	$0X0\$$
11	$X0\$$
11	$10\$$
1	$0\$$
backtrack	
0011	$X1\$$

input	stack
0011	$11\$$
backtrack	
0011	$0X1\$$
011	$X1\$$
011	$11\$$
backtrack	
011	$0X1\$$
11	$X1\$$
11	$11\$$

LL-parsing: example

S → X0

S → X1

X → 1

X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$
backtrack	
0011	$0X0\$$
011	$X0\$$
011	$10\$$
backtrack	
011	$0X0\$$
11	$X0\$$
11	$10\$$
1	$0\$$
backtrack	
0011	$X1\$$

input	stack
0011	$11\$$
backtrack	
0011	$0X1\$$
011	$X1\$$
011	$11\$$
backtrack	
011	$0X1\$$
11	$X1\$$
11	$11\$$
1	$1\$$

LL-parsing: example

S → X0

S → X1

X → 1

X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$
backtrack	
0011	$0X0\$$
011	$X0\$$
011	$10\$$
backtrack	
011	$0X0\$$
11	$X0\$$
11	$10\$$
1	$0\$$
backtrack	
0011	$X1\$$

input	stack
0011	$11\$$
backtrack	
0011	$0X1\$$
011	$X1\$$
011	$11\$$
backtrack	
011	$0X1\$$
11	$X1\$$
11	$11\$$
1	$1\$$
ϵ	$\$$

Success!

Question 1

Given the grammar shown here, assume an LL-parser that, when it has a choice about which rule to predict, will first try the topmost one it hasn't tried yet in the given list. How many times will this parser backtrack before accepting the NP *Dutch cheese*?

NP $\rightarrow$ N
NP $\rightarrow$ A NP
NP $\rightarrow$ NP N
N $\rightarrow$ cheese
N $\rightarrow$ lover
A $\rightarrow$ Dutch

a. 0

b. 1

c. 2

d. 3 or more

Question 1

Given the grammar shown here, assume an LL-parser that, when it has a choice about which rule to predict, will first try the topmost one it hasn't tried yet in the given list. How many times will this parser backtrack before accepting the NP *Dutch cheese*?

NP $\rightarrow$ N
NP $\rightarrow$ A NP
NP $\rightarrow$ NP N
N $\rightarrow$ cheese
N $\rightarrow$ lover
A $\rightarrow$ Dutch

a. 0

b. 1

c. 2

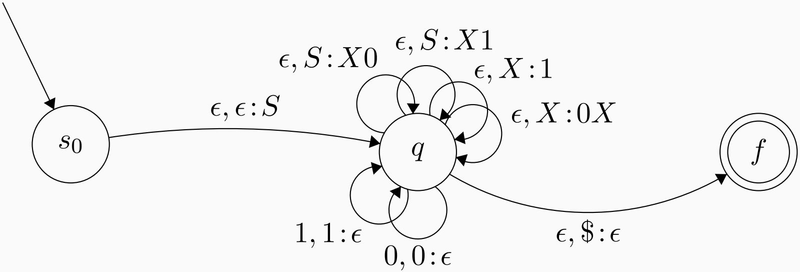
d. 3 or more

LL-parser as pushdown automaton

$S \to X0 \qquad S \to X1 \qquad X \to 1 \qquad X \to 0X$

LL-parser as pushdown automaton

S → X0 S → X1 X → 1 X → 0X



Lookahead for LL-parsing

S $\to$ X0

S $\to$ X1

X $\to$ 1

X $\to$ 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$

backtrack

...

Lookahead for LL-parsing

S $\rightarrow$ X0

S $\rightarrow$ X1

X $\rightarrow$ 1

X $\rightarrow$ 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$

backtrack

...

- Problem: Lots of backtracking is inefficient!

Lookahead for LL-parsing

S $\rightarrow$ X0

S $\rightarrow$ X1

X $\rightarrow$ 1

X $\rightarrow$ 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$

backtrack

...

- Problem: Lots of backtracking is inefficient!
- Can improve efficiency by paying attention to what is coming later in input: **lookahead**

Lookahead for LL-parsing

S → X0

S → X1

X → 1

X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$

backtrack

...

- Problem: Lots of backtracking is inefficient!
- Can improve efficiency by paying attention to what is coming later in input: **lookahead**
- LL(k): grammars that on the basis of knowing the following k symbols in the input always make the right choice in applying rules (=no backtracking)

Lookahead for LL-parsing

S → X0
S → X1
X → 1
X → 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$

backtrack

...

- Problem: Lots of backtracking is inefficient!
- Can improve efficiency by paying attention to what is coming later in input: **lookahead**
- LL(k): grammars that on the basis of knowing the following k symbols in the input always make the right choice in applying rules (=no backtracking)
- ex: top-down LL(1) = parsing made solely on the basis of the next symbol to be read
- **Parse table**: assigns production rule choice to *combination* of top of the stack + following k symbols

Lookahead for LL-parsing

S $\rightarrow$ X0

S $\rightarrow$ X1

X $\rightarrow$ 1

X $\rightarrow$ 0X

input	stack
0011	$\$$
0011	$S\$$
0011	$X0\$$
0011	$10\$$

backtrack

...

- Problem: Lots of backtracking is inefficient!
- Can improve efficiency by paying attention to what is coming later in input: **lookahead**
- LL(k): grammars that on the basis of knowing the following k symbols in the input always make the right choice in applying rules (=no backtracking)

	0	1
S	$S \rightarrow X0$	$S \rightarrow X0$
	$S \rightarrow X1$	$S \rightarrow X1$
X	$X \rightarrow 0X$	$X \rightarrow 1$

Left-recursion

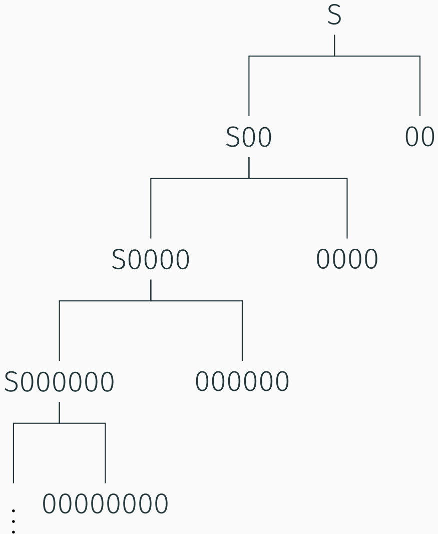
Warning: LL-parsing of *left-recursive* structures can fail to terminate:

Left-recursion

Warning: LL-parsing of *left-recursive* structures can fail to terminate:

$S \rightarrow S00$

$S \rightarrow 00$

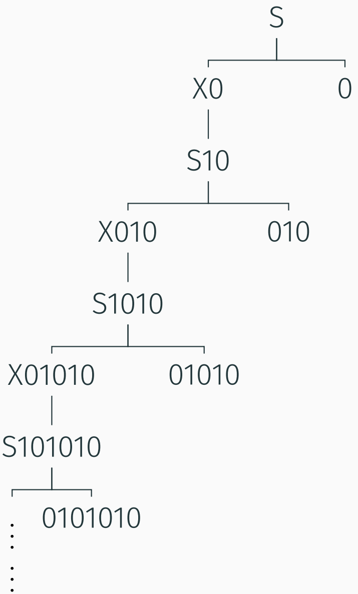


Left-recursion

$S \to X0$

$S \to 0$

$X \to S1$



Question 2

Dutch cheese lover has two readings (=distinct interpretations) in the same way as *Nederlandse kaasliefhebber*. Given the same grammar and LL-parser as the previous question, which readings corresponds to the parse tree generated for *Dutch cheese lover*? (Hint: Think about constituency!)

$NP \rightarrow N$
$NP \rightarrow A\ NP$
$NP \rightarrow NP\ N$
$N \rightarrow$ cheese
$N \rightarrow$ lover
$A \rightarrow$ Dutch

1. Lover of Dutch cheese
2. Dutch person who loves cheese
3. The parser will not terminate given this input

Question 2

Dutch cheese lover has two analogous readings (=distinct interpretations) to *Nederlandse kaasliefhebber*. Given the same grammar and LL-parser as the previous question, which readings corresponds to the parse tree generated for *Dutch cheese lover*? (Hint: Think about constituency!)

$$
\begin{aligned}
NP &\rightarrow N \\
NP &\rightarrow A\ NP \\
NP &\rightarrow NP\ N \\
N &\rightarrow \text{cheese} \\
N &\rightarrow \text{lover} \\
A &\rightarrow \text{Dutch}
\end{aligned}
$$

1. Lover of Dutch cheese
2. **Dutch person who loves cheese**
3. The parser will not terminate given this input

Bottom-up (‘shift-reduce’) parsing

Bottom-up parsing

Basic idea: Starting from the terminal symbols, apply rules 'backwards' until you get to the start symbol

Bottom-up parsing

Basic idea: Starting from the terminal symbols, apply rules ‘backwards’ until you get to the start symbol

1. Begin with an empty stack.

Bottom-up parsing

Basic idea: Starting from the terminal symbols, apply rules ‘backwards’ until you get to the start symbol

1. Begin with an empty stack.
2. Do one of the following:
   a) **Shift**: Read a symbol from the input and put it on top of the stack

Bottom-up parsing

Basic idea: Starting from the terminal symbols, apply rules 'backwards' until you get to the start symbol

1. Begin with an empty stack.
2. Do one of the following:
   a) **Shift**: Read a symbol from the input and put it on top of the stack
   b) **Reduce**: replace α on top of the stack with X if there exists a production rule $X \to \alpha$

Bottom-up parsing

Basic idea: Starting from the terminal symbols, apply rules 'backwards' until you get to the start symbol

1. Begin with an empty stack.
2. Do one of the following:
   a) **Shift**: Read a symbol from the input and put it on top of the stack
   b) **Reduce**: replace α on top of the stack with X if there exists a production rule $X \rightarrow \alpha$
3. Repeat 3 until either:
   a) No more input to read and stack consists only of S (accept)
   b) Input fully read but stack non-empty: Go back to most recent choice point and make a different choice (backtrack)

Bottom-up parsing: Example

$S \rightarrow X0$

$S \rightarrow 0$

$X \rightarrow S1$

input	stack	operation
010		
10	0	shift
10	S	reduce
0	S1	shift
0	X	reduce
	X0	shift
	S	reduce
	S	accept

Left-corner parsing

- Top-down & bottom-up parsing both explicit algorithms to derive strings given a grammar
- Top-down: ‘What strings should the grammar generate?’
- Bottom-up: ‘What strings does the grammar recognize?’

Left-corner parsing

- Top-down & bottom-up parsing both explicit algorithms to derive strings given a grammar
- Top-down: ‘What strings should the grammar generate?’
- Bottom-up: ‘What strings does the grammar recognize?’

But both kinds of parsing have **weaknesses**:

Left-corner parsing

- Top-down & bottom-up parsing both explicit algorithms to derive strings given a grammar
- Top-down: ‘What strings should the grammar generate?’
- Bottom-up: ‘What strings does the grammar recognize?’

But both kinds of parsing have **weaknesses**:

- Top-down: Makes predictions without checking whether the input fits
- Bottom-up: Doesn’t check whether the structures it builds correspond to what might be predicted starting from S

Left-corner parsing

- Top-down & bottom-up parsing both explicit algorithms to derive strings given a grammar
- Top-down: 'What strings should the grammar generate?'
- Bottom-up: 'What strings does the grammar recognize?'

But both kinds of parsing have **weaknesses**:

- Top-down: Makes predictions without checking whether the input fits
- Bottom-up: Doesn't check whether the structures it builds correspond to what might be predicted starting from S

These properties are complementary–can we get the best of both worlds?

Left-corner parsing

Basic idea: Do a bottom-up parse, but use top-down parsing to make some predictions and constrain the search space

Left-corner parsing

Basic idea: Do a bottom-up parse, but use top-down parsing to make some predictions and constrain the search space

Approach (informal):

- For production rule $X \rightarrow A\ B\ C\ D...$, the first element on the righthand side of the arrow is the '**left corner**'

Left-corner parsing

Basic idea: Do a bottom-up parse, but use top-down parsing to make some predictions and constrain the search space

Approach (informal):

- For production rule X $\rightarrow$ A B C D... , the first element on the righthand side of the arrow is the '**left corner**'
- Commit to parse of A, then make *predictions* about what larger structures it will be a part of

$\star$ Gets around left-recursion problem of LL-parsing!

Left-corner parsing: more formal

1. Start with a stack with $ on top
2. Put **expectation** [S] of start symbol S on top of stack

Left-corner parsing: more formal

1. Start with a stack with $ on top
2. Put **expectation** [S] of start symbol S on top of stack
3. Do one of the following:
   - **Shift**: Take current symbol from input and place on top of stack
   - **Announce**: Replace Z on top of the stack with X if there exists a production rule $X \rightarrow Z$
   - **Announce**: Replace Z on top of stack with Y then $[\beta]$ if there exists a production rule $Y \rightarrow Z\beta$, where β is a sequence of 1 or more symbols.

Left-corner parsing: more formal

1. Start with a stack with $ on top
2. Put **expectation** [S] of start symbol S on top of stack
3. Do one of the following:
   - **Shift**: Take current symbol from input and place on top of stack
   - **Announce**: Replace Z on top of the stack with X if there exists a production rule $X \rightarrow Z$
   - **Announce**: Replace Z on top of stack with Y then $[\beta]$ if there exists a production rule $Y \rightarrow Z\beta$, where β is a sequence of 1 or more symbols.
   - **Match**: If the topmost element of the stack is Z and the secondmost element is $[Z]$, remove both from the stack.

Left-corner parsing: more formal

1. Start with a stack with $ on top
2. Put **expectation** [S] of start symbol S on top of stack
3. Do one of the following:
    - **Shift**: Take current symbol from input and place on top of stack
    - **Announce**: Replace Z on top of the stack with X if there exists a production rule $X \to Z$
    - **Announce**: Replace Z on top of stack with Y then $[\beta]$ if there exists a production rule $Y \to Z\beta$, where β is a sequence of 1 or more symbols.
    - **Match**: If the topmost element of the stack is Z and the secondmost element is $[Z]$, remove both from the stack.
4. Repeat step 3 there is nothing left in the input to read and only $ is left in the stack. Backtrack if necessary.

Left-corner parsing: An example

(Assumption: DP is the start symbol)

$$
\begin{aligned}
\text{DP} &\rightarrow \text{D NP} \\
\text{NP} &\rightarrow \text{N} \\
\text{NP} &\rightarrow \text{A N} \\
\text{V} &\rightarrow \text{bakt} \\
\text{D} &\rightarrow \text{de} \mid \text{een} \\
\text{N} &\rightarrow \text{man} \mid \text{taart} \\
\text{A} &\rightarrow \text{lekkere}
\end{aligned}
$$

Left-corner parsing: An example

(Assumption: DP is the start symbol)

input buffer	stack	operation

$$
\begin{aligned}
DP &\rightarrow D\ NP \\
NP &\rightarrow N \\
NP &\rightarrow A\ N \\
V &\rightarrow \text{bakt} \\
D &\rightarrow \text{de} \mid \text{een} \\
N &\rightarrow \text{man} \mid \text{taart} \\
A &\rightarrow \text{lekkere}
\end{aligned}
$$

Left-corner parsing: An example

(Assumption: DP is the start symbol)

$$\begin{aligned}
DP &\rightarrow D\ NP \\
NP &\rightarrow N \\
NP &\rightarrow A\ N \\
V &\rightarrow \text{bakt} \\
D &\rightarrow \text{de} \mid \text{een} \\
N &\rightarrow \text{man} \mid \text{taart} \\
A &\rightarrow \text{lekkere}
\end{aligned}$$

input buffer	stack	operation
een lekkere taart	$	

Left-corner parsing: An example

(Assumption: DP is the start symbol)

$$\begin{aligned}
DP &\rightarrow D\ NP \\
NP &\rightarrow N \\
NP &\rightarrow A\ N \\
V &\rightarrow \text{bakt} \\
D &\rightarrow \text{de} \mid \text{een} \\
N &\rightarrow \text{man} \mid \text{taart} \\
A &\rightarrow \text{lekkere}
\end{aligned}$$

input buffer	stack	operation
een lekkere taart	$	
een lekkere taart	$ [DP]	

Left-corner parsing: An example

(Assumption: DP is the start symbol)

$$
\begin{aligned}
DP &\rightarrow D\ NP \\
NP &\rightarrow N \\
NP &\rightarrow A\ N \\
V &\rightarrow \text{bakt} \\
D &\rightarrow \text{de} \mid \text{een} \\
N &\rightarrow \text{man} \mid \text{taart} \\
A &\rightarrow \text{lekkere}
\end{aligned}
$$

input buffer	stack	operation
een lekkere taart	$	
een lekkere taart	$ [DP]	
lekkere taart	$ [DP] een	shift

Left-corner parsing: An example

(Assumption: DP is the start symbol)

$DP \rightarrow D\ NP$

$NP \rightarrow N$

$NP \rightarrow A\ N$

$V \rightarrow$ bakt

$D \rightarrow$ de | een

$N \rightarrow$ man | taart

$A \rightarrow$ lekkere

input buffer	stack	operation
een lekkere taart	$	
een lekkere taart	$ [DP]	
lekkere taart	$ [DP] een	shift
lekkere taart	$ [DP] D	announce

Left-corner parsing: An example

(Assumption: DP is the start symbol)

$$
\begin{aligned}
DP &\rightarrow D\ NP \\
NP &\rightarrow N \\
NP &\rightarrow A\ N \\
V &\rightarrow \text{bakt} \\
D &\rightarrow \text{de} \mid \text{een} \\
N &\rightarrow \text{man} \mid \text{taart} \\
A &\rightarrow \text{lekkere}
\end{aligned}
$$

input buffer	stack	operation
een lekkere taart	$	
een lekkere taart	$ [DP]	
lekkere taart	$ [DP] een	shift
lekkere taart	$ [DP] D	announce
lekkere taart	$ [DP] DP [NP]	announce

Left-corner parsing: An example

(Assumption: DP is the start symbol)

$$
\begin{aligned}
DP &\rightarrow D\ NP \\
NP &\rightarrow N \\
NP &\rightarrow A\ N \\
V &\rightarrow \text{bakt} \\
D &\rightarrow \text{de} \mid \text{een} \\
N &\rightarrow \text{man} \mid \text{taart} \\
A &\rightarrow \text{lekkere}
\end{aligned}
$$

input buffer	stack	operation
een lekkere taart	$	
een lekkere taart	$ [DP]	
lekkere taart	$ [DP] een	shift
lekkere taart	$ [DP] D	announce
lekkere taart	$ [DP] DP [NP]	announce
taart	$ [DP] DP [NP] lekkere	shift

Left-corner parsing: An example

(Assumption: DP is the start symbol)

$DP \rightarrow D\ NP$

$NP \rightarrow N$

$NP \rightarrow A\ N$

$V \rightarrow \text{bakt}$

$D \rightarrow \text{de} \mid \text{een}$

$N \rightarrow \text{man} \mid \text{taart}$

$A \rightarrow \text{lekkere}$

input buffer	stack	operation
een lekkere taart	$	
een lekkere taart	$ [DP]	
lekkere taart	$ [DP] een	shift
lekkere taart	$ [DP] D	announce
lekkere taart	$ [DP] DP [NP]	announce
taart	$ [DP] DP [NP] lekkere	shift
taart	$ [DP] DP [NP] A	announce

Left-corner parsing: An example

(Assumption: DP is the start symbol)

$DP \rightarrow D\ NP$

$NP \rightarrow N$

$NP \rightarrow A\ N$

$V \rightarrow$ bakt

$D \rightarrow$ de | een

$N \rightarrow$ man | taart

$A \rightarrow$ lekkere

input buffer	stack	operation
een lekkere taart	$	
een lekkere taart	$ [DP]	
lekkere taart	$ [DP] een	shift
lekkere taart	$ [DP] D	announce
lekkere taart	$ [DP] DP [NP]	announce
taart	$ [DP] DP [NP] lekkere	shift
taart	$ [DP] DP [NP] A	announce
taart	$ [DP] DP [NP] NP [N]	announce

Left-corner parsing: An example

(Assumption: DP is the start symbol)

$DP \rightarrow D\ NP$

$NP \rightarrow N$

$NP \rightarrow A\ N$

$V \rightarrow bakt$

$D \rightarrow de \mid een$

$N \rightarrow man \mid taart$

$A \rightarrow lekkere$

input buffer	stack	operation
een lekkere taart	$	
een lekkere taart	$ [DP]	
lekkere taart	$ [DP] een	shift
lekkere taart	$ [DP] D	announce
lekkere taart	$ [DP] DP [NP]	announce
taart	$ [DP] DP [NP] lekkere	shift
taart	$ [DP] DP [NP] A	announce
taart	$ [DP] DP [NP] NP [N]	announce
	$ [DP] DP [NP] NP [N] taart	shift

Left-corner parsing: An example

(Assumption: DP is the start symbol)

$DP \rightarrow D\ NP$

$NP \rightarrow N$

$NP \rightarrow A\ N$

$V \rightarrow$ bakt

$D \rightarrow$ de | een

$N \rightarrow$ man | taart

$A \rightarrow$ lekkere

input buffer	stack	operation
een lekkere taart	$	
een lekkere taart	$ [DP]	
lekkere taart	$ [DP] een	shift
lekkere taart	$ [DP] D	announce
lekkere taart	$ [DP] DP [NP]	announce
taart	$ [DP] DP [NP] lekkere	shift
taart	$ [DP] DP [NP] A	announce
taart	$ [DP] DP [NP] NP [N]	announce
	$ [DP] DP [NP] NP [N] taart	shift
	$ [DP] DP [NP] NP [N] N	announce

Left-corner parsing: An example

(Assumption: DP is the start symbol)

$DP \rightarrow D\ NP$

$NP \rightarrow N$

$NP \rightarrow A\ N$

$V \rightarrow bakt$

$D \rightarrow de \mid een$

$N \rightarrow man \mid taart$

$A \rightarrow lekkere$

input buffer	stack	operation
een lekkere taart	$	
een lekkere taart	$ [DP]	
lekkere taart	$ [DP] een	shift
lekkere taart	$ [DP] D	announce
lekkere taart	$ [DP] DP [NP]	announce
taart	$ [DP] DP [NP] lekkere	shift
taart	$ [DP] DP [NP] A	announce
taart	$ [DP] DP [NP] NP [N]	announce
	$ [DP] DP [NP] NP [N] taart	shift
	$ [DP] DP [NP] NP [N] N	announce
	$ [DP] DP [NP] NP	match N

Left-corner parsing: An example

(Assumption: DP is the start symbol)

$$DP \to D\ NP$$
$$NP \to N$$
$$NP \to A\ N$$
$$V \to bakt$$
$$D \to de \mid een$$
$$N \to man \mid taart$$
$$A \to lekkere$$

input buffer	stack	operation
een lekkere taart	$	
een lekkere taart	$ [DP]	
lekkere taart	$ [DP] een	shift
lekkere taart	$ [DP] D	announce
lekkere taart	$ [DP] DP [NP]	announce
taart	$ [DP] DP [NP] lekkere	shift
taart	$ [DP] DP [NP] A	announce
taart	$ [DP] DP [NP] NP [N]	announce
	$ [DP] DP [NP] NP [N] taart	shift
	$ [DP] DP [NP] NP [N] N	announce
	$ [DP] DP [NP] NP	match N
	$ [DP] DP	match NP

Left-corner parsing: An example

(Assumption: DP is the start symbol)

$DP \to D\ NP$

$NP \to N$

$NP \to A\ N$

$V \to bakt$

$D \to de \mid een$

$N \to man \mid taart$

$A \to lekkere$

input buffer	stack	operation
een lekkere taart	$	
een lekkere taart	$ [DP]	
lekkere taart	$ [DP] een	shift
lekkere taart	$ [DP] D	announce
lekkere taart	$ [DP] DP [NP]	announce
taart	$ [DP] DP [NP] lekkere	shift
taart	$ [DP] DP [NP] A	announce
taart	$ [DP] DP [NP] NP [N]	announce
	$ [DP] DP [NP] NP [N] taart	shift
	$ [DP] DP [NP] NP [N] N	announce
	$ [DP] DP [NP] NP	match N
	$ [DP] DP	match NP
	$	match DP

Question 3

Humans also need to parse sentences in order to understand them, since knowing the structure of a sentence is required to interpret it. Which parsing method seems most similar to how humans parse sentences?

1. Top-down parsing
2. Bottom-up parsing
3. Left-corner parsing

One last parsing method

Cocke Younger Kasami (CYK / CKY) parsing

- Intuition: List out every possible way of combining every consecutive group of symbols in string
- Constraint: CFG must be in the 'Chomsky normal form'

$$\text{A} \rightarrow \text{B C}$$
$$\text{A} \rightarrow \text{a}$$

- Let i be the length of the input:

```
for n in range 1 to i:
    For every combination of length n:
        find a rule that allows
        that combination and store
        the transformation
```

CYK: Simple example

String: abc

$S \rightarrow X\ B$

$S \rightarrow A\ Z$

$X \rightarrow a$

$Y \rightarrow b$

$Z \rightarrow c$

$B \rightarrow Y\ Z$

$A \rightarrow X\ Y$

CYK: Simple example

String: abc

$S \rightarrow X\ B$

$S \rightarrow A\ Z$

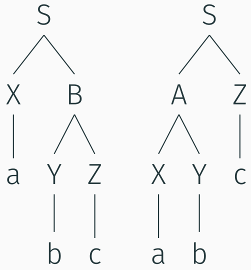
$X \rightarrow a$

$Y \rightarrow b$

$Z \rightarrow c$

$B \rightarrow Y\ Z$

$A \rightarrow X\ Y$



CYK: Simple example

String: abc

$S \rightarrow X\ B$

$S \rightarrow A\ Z$

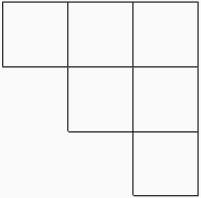
$X \rightarrow a$

$Y \rightarrow b$

$Z \rightarrow c$

$B \rightarrow Y\ Z$

$A \rightarrow X\ Y$



Start: Empty table

CYK: Simple example

String: abc

$S \rightarrow X\ B$

$S \rightarrow A\ Z$

$X \rightarrow a$

$Y \rightarrow b$

$Z \rightarrow c$

$B \rightarrow Y\ Z$

$A \rightarrow X\ Y$

X		
	Y	
		Z

Sequences of length 1:
a, b, c
Left to right in bottom cells

CYK: Simple example

String: abc

$S \rightarrow X\ B$

$S \rightarrow A\ Z$

$X \rightarrow a$

$Y \rightarrow b$

$Z \rightarrow c$

$B \rightarrow Y\ Z$

$A \rightarrow X\ Y$

X	A	
	Y	B
		Z

Sequences of length 2:
ab, bc

CYK: Simple example

String: abc

$S \to X\ B$

$S \to A\ Z$

$X \to a$

$Y \to b$

$Z \to c$

$B \to Y\ Z$

$A \to X\ Y$

X	A	S
	Y	B
		Z

Sequences of length 3:

abc

CYK parsing procedure

- Walk through the parse table

CYK parsing procedure

- Walk through the parse table
- Begin with all cells for length 1, then the cells for length 2, and so on

CYK parsing procedure

- Walk through the parse table
- Begin with all cells for length 1, then the cells for length 2, and so on
- For each cell c:
  - Look at all cells to the **left** and **below** of c to see if there are rules whose righthand side is nth element to the left of c followed by nth element from the bottom below c

CYK parsing procedure

- Walk through the parse table
- Begin with all cells for length 1, then the cells for length 2, and so on
- For each cell c:
    - Look at all cells to the **left** and **below** of c to see if there are rules whose righthand side is nth element to the left of c followed by nth element from the bottom below c
    - If yes, put the left side of the arrow for that rule in c

CYK parsing procedure

- Walk through the parse table
- Begin with all cells for length 1, then the cells for length 2, and so on
- For each cell c:
  - Look at all cells to the **left** and **below** of c to see if there are rules whose righthand side is nth element to the left of c followed by nth element from the bottom below c
  - If yes, put the left side of the arrow for that rule in c
  - Possible that there are multiple rules which fit these criteria: list them all

CYK parsing procedure

- Walk through the parse table
- Begin with all cells for length 1, then the cells for length 2, and so on
- For each cell c:
    - Look at all cells to the **left** and **below** of c to see if there are rules whose righthand side is nth element to the left of c followed by nth element from the bottom below c
    - If yes, put the left side of the arrow for that rule in c
    - Possible that there are multiple rules which fit these criteria: list them all
- Parse successful if we can put something (start symbol) in the top right corner

CYK - more complex example

$$\begin{aligned}
S &\rightarrow \text{N VP} \\
N &\rightarrow \text{ADJ N} \\
N &\rightarrow \text{N N} \\
N &\rightarrow \text{vissen} \\
N &\rightarrow \text{kommen} \\
\text{ADJ} &\rightarrow \text{hongerige} \\
\text{VP} &\rightarrow \text{V PP} \\
V &\rightarrow \text{vissen} \\
\text{PP} &\rightarrow \text{P N} \\
P &\rightarrow \text{in}
\end{aligned}$$

String: hongerige vissen vissen in vissen kommen

CYK - more complex example

$S \rightarrow N\ VP$

$N \rightarrow ADJ\ N$

$N \rightarrow N\ N$

$N \rightarrow$ vissen

$N \rightarrow$ kommen

$ADJ \rightarrow$ hongerige

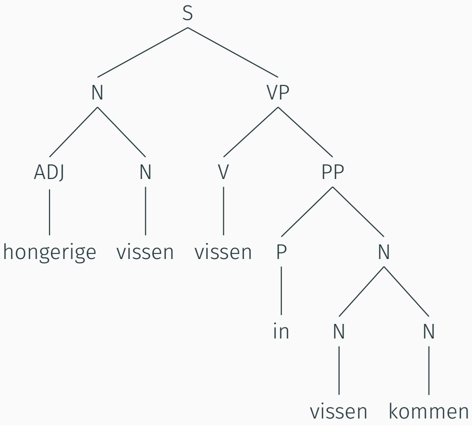
$VP \rightarrow V\ PP$

$V \rightarrow$ vissen

$PP \rightarrow P\ N$

$P \rightarrow$ in

String: hongerige vissen vissen in vissen kommen



CYK - more complex example

S → N VP
N → ADJ N
N → N N
N → vissen
N → kommen
ADJ → hongerige
VP → V PP
V → vissen
PP → P N
P → in

String: hongerige vissen vissen in vissen kommen

empty table for 6 terminals.

CYK - more complex example

S → N VP
N → ADJ N
N → N N
N → vissen
N → kommen
ADJ → hongerige
VP → V PP
V → vissen
PP → P N
P → in

ADJ					
	N V				
		N V			
			P		
				N V	
					N

String: hongerige vissen vissen in vissen kommen

length: 1

CYK - more complex example

$$
\begin{aligned}
S &\rightarrow N\ VP \\
N &\rightarrow ADJ\ N \\
N &\rightarrow N\ N \\
N &\rightarrow \text{vissen} \\
N &\rightarrow \text{kommen} \\
ADJ &\rightarrow \text{hongerige} \\
VP &\rightarrow V\ PP \\
V &\rightarrow \text{vissen} \\
PP &\rightarrow P\ N \\
P &\rightarrow \text{in}
\end{aligned}
$$

ADJ	N				
	N V	N			
		N V	-		
			P	PP	
				N V	N
					N

String: hongerige vissen vissen in vissen kommen

length: 2

CYK - more complex example

$S \rightarrow N\ VP$
$N \rightarrow ADJ\ N$
$N \rightarrow N\ N$
$N \rightarrow \text{vissen}$
$N \rightarrow \text{kommen}$
$ADJ \rightarrow \text{hongerige}$
$VP \rightarrow V\ PP$
$V \rightarrow \text{vissen}$
$PP \rightarrow P\ N$
$P \rightarrow \text{in}$

ADJ	N	N N			
	N V	N	-		
		N V	-	VP	
			P	PP	PP
				N V	N
					N

String: hongerige vissen vissen in vissen kommen

length: 3

CYK - more complex example

$S \rightarrow N\ VP$
$N \rightarrow ADJ\ N$
$N \rightarrow N\ N$
$N \rightarrow$ vissen
$N \rightarrow$ kommen
$ADJ \rightarrow$ hongerige
$VP \rightarrow V\ PP$
$V \rightarrow$ vissen
$PP \rightarrow P\ N$
$P \rightarrow$ in

ADJ	N	N N	-		
	N V	N	-	S	
		N V	-	VP	VP
			P	PP	PP
				N V	N
					N

String: hongerige vissen vissen in vissen kommen

length: 4

CYK - more complex example

$S \rightarrow N\ VP$

$N \rightarrow ADJ\ N$

$N \rightarrow N\ N$

$N \rightarrow$ vissen

$N \rightarrow$ kommen

$ADJ \rightarrow$ hongerige

$VP \rightarrow V\ PP$

$V \rightarrow$ vissen

$PP \rightarrow P\ N$

$P \rightarrow$ in

ADJ	N	N N	-	-	
	N V	N	-	S	S
		N V	-	VP	VP
			P	PP	PP
				N V	N
					N

String: hongerige vissen vissen in vissen kommen

length: 5

CYK - more complex example

$S \rightarrow N\ VP$
$N \rightarrow ADJ\ N$
$N \rightarrow N\ N$
$N \rightarrow$ vissen
$N \rightarrow$ kommen
$ADJ \rightarrow$ hongerige
$VP \rightarrow V\ PP$
$V \rightarrow$ vissen
$PP \rightarrow P\ N$
$P \rightarrow$ in

ADJ	N	N N	-	-	S
	N V	N	-	S	S
		N V	-	VP	VP
			P	PP	PP
				N V	N
					N

String: hongerige vissen vissen in vissen kommen

length: 6

Question 4

Given this grammar, what symbol goes in the cell labeled 1 for the CYK parse of een lekkere taart op een grote tafel? (The bottom cells have been filled in for you.)

D						
	A					
		N				1
			P			
				D		
					A	
						N

$\mathrm{DP} \rightarrow \mathrm{D\ NP}$
$\mathrm{PP} \rightarrow \mathrm{P\ DP}$
$\mathrm{NP} \rightarrow \mathrm{A\ N}$
$\mathrm{NP} \rightarrow \mathrm{A\ NP}$
$\mathrm{NP} \rightarrow \mathrm{N\ PP}$
$\mathrm{P} \rightarrow \mathrm{op}$
$\mathrm{D} \rightarrow \mathrm{een}$
$\mathrm{N} \rightarrow \mathrm{tafel} \mid \mathrm{taart}$
$\mathrm{A} \rightarrow \mathrm{grote}$
$\mathrm{A} \rightarrow \mathrm{lekkere}$

Question 4

Given this grammar, what symbol goes in the cell labeled **1** for the CYK parse of een lekkere taart op een grote tafel? (The bottom cells have been filled in for you.)

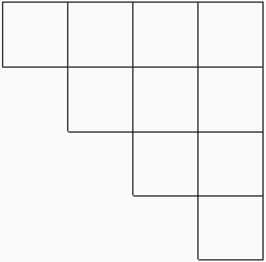
DP → D NP
PP → P DP
NP → A N
NP → A NP
NP → N PP
P → op
D → een
N → tafel | taart
A → grote
A → lekkere

D	-	DP	-	-	-	DP
	A	NP	-	-	-	NP
		N	-	-	-	**NP**
			P	-	-	PP
				D	-	DP
					A	NP
						N

Parsing methods and time

Parsing algorithms vary in how long they take.

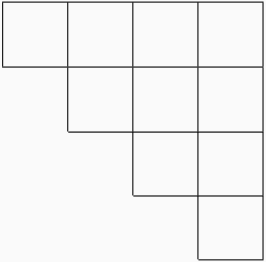
How many steps does CYK take?



Parsing methods and time

Parsing algorithms vary in how long they take.

How many steps does CYK take?



- 4 words: 10 steps (besides length 1)
- 5 words: 20 steps (besides length 1)
- 6 words: 35 steps (besides length 1)
- Big-O notation: time as a function of length n (ignoring constants): $\mathcal{O}(n^3)$

Summary

- Several deterministic parsing methods: top-down, bottom-up, (left-corner), CYK
  - Different strengths and weaknesses
- LL-parsing: top-down, can be inefficient or fail to terminate on left-recursive structures; enhance with lookahead (LL(k))
- CYK